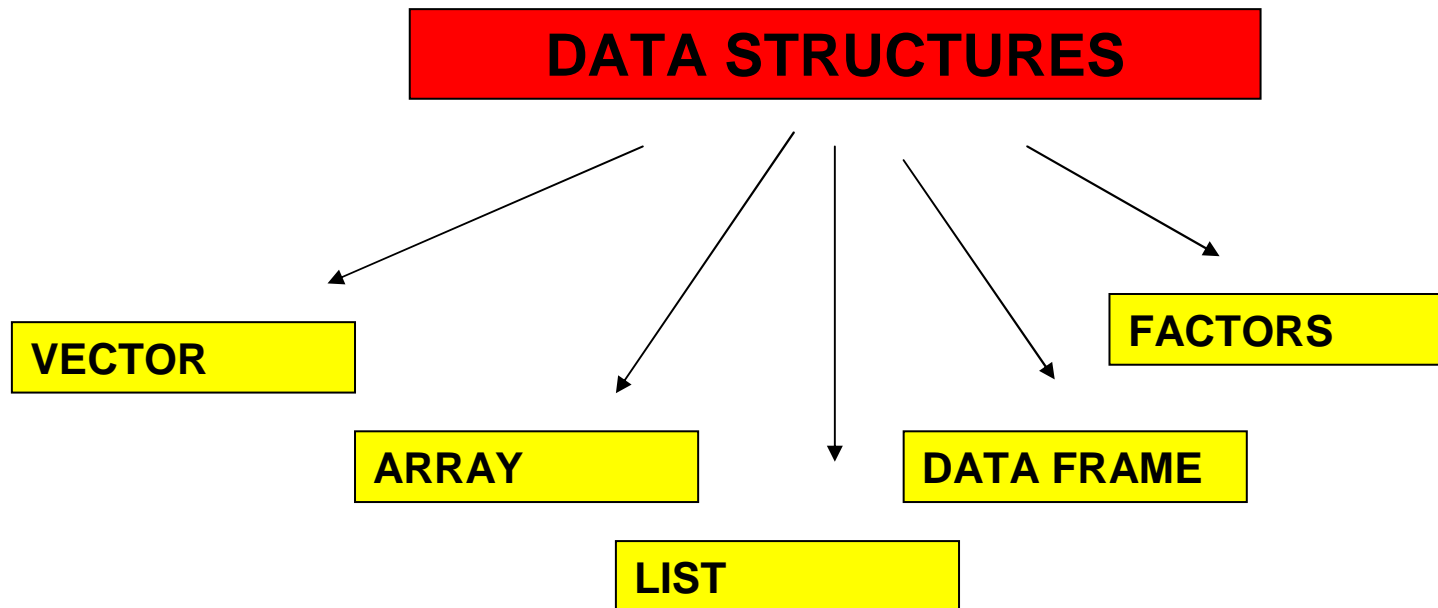

Data structures

- Vectors
- Matrices and arrays
- Lists
- Data frames

Data structures

Each object created in R can be referred to one of the following *data structures*.



The simplest data structure is the vector

Vectors

A vector is an entity consisting of an ordered collection of elements all of the same type or “*mode*”.

The “mode” of an object is the basic type of its fundamental constituents.

After the “mode” of its elements, a vector can be:

- Numeric
 - Logical
 - Character
 - Complex
-

Numeric Vectors

The numeric vector is “a single entity consisting of an ordered collection of *numbers*”

To input in the R workspace a vector named “x” whose elements are (0.5, 3.1, 2.2, 4) type:

```
x <- c (0.5, 3.1, 2.2, 4)
```

where “<-” is the assignment operator

and **c()** is the concatenation function whose output is a vector obtained by concatenating its arguments end to end.

A numeric variable is considered itself a vector of length one.

Vector Arithmetic

It is possible to calculate arithmetical expression using vectors, for example:

```
a<- c(1,2,3)
```

```
b<- c(0.1, 0.2, 0.3)
```

```
c<- a+b; d<- a*b; e <- b^a
```

Vectors occurring in the same expression may have different lengths:

- The output will have the same length as the longest vector
- Shorter vectors in the expression will be **recycled** until they match the length of the longest vector

Example:

```
x <- c(1,2,3,4,5);    y <- c(1,2);    z<- x + y -2
```

Generating regular sequences

R provides some functions to generate regular sequences of numbers:

seq(from=,to=,by=, length=)

- Either “by” or “length” need to be provided
- If neither is provided, the default is “by”=1

Example: `x<-seq(1,5)` gives the vector `x=(1,2,3,4,5)`

N.B.: the same results is obtained using the colon “:”

`x<-1:5`

while `x <- 5:1` generates the sequence backwards.

Another function to replicate objects in many different ways is:

rep(arg=, times=, each=, len=,)

Example:

```
> rep(1:4, times=2)
```

```
[1] 1 2 3 4 1 2 3 4
```

```
> rep(1:4, each = 2)
```

```
>[1] 1 1 2 2 3
```

```
> rep(1:4, each = 2, len = 4) # first 4 only.
```

```
[1] 1 1 2 2
```

```
> rep(1:4, each = 2, len = 10) # 8 integers plus two recycled 1's.
```

```
[1] 1 1 2 2 3 3 4 4 1 1
```

Logical vectors

A logical vector is generated by logical expressions:

```
> x<- 1:10
```

```
> y<- x>3
```

```
> y
```

```
FALSE FALSE FALSE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
```

Its elements can assume the value TRUE or FALSE (or NA if the value is Not Available).

Logical vectors can be used in arithmetic expressions, where they are coerced into numeric vectors, **FALSE** becoming **0** and **TRUE** becoming **1**.

Character vectors

- Characters vectors are frequently used in the customization of the plots;
- They are created with the concatenation function `c()`;
- Their elements are entered using either double (“) or single (‘) quotes.

Example:

```
a<- c(“Hello”, “everybody!”)
```

Complex vectors

Complex vectors can be created with the function **complex()**.

The vector can be specified either by giving its length, its real and imaginary parts, or modulus and argument.

Z <- complex (length.out = , real = numeric(), imaginary = numeric(), modulus = , argument =)

Length.out: numeric, desired length of the output vector, inputs being recycled as needed.

Real: numeric vector

Imaginary: numeric vector

Modulus: numeric vector

Argument: numeric vector

Index vectors

Subsets of the elements of a vector may be selected by appending to the name of the vector an *index vector* in square brackets.

Here will be presented three types of index vectors:

1. **Vector of positive integer quantities:**

the values in the index vector have to be in the interval [1, length(vector)]

Example:

x[n] will select the element of place n

x[c(2,5,7)] will select the elements of place 2,5 and 7

x[1:10] will select the first 10 elements of x
(it is assumed that length(x) \geq 10)

2. **Vector of negative integer quantities:**

it identifies the values to be excluded from the selection

Example:

`y <- x[-(1:5)]`

will give all but the first five elements of x

`y <- x[-c(2,5,7)]`
7th

will give all but the 2nd,5th and 7th elements of x

3. **Logical vector:**

the elements of the vector which satisfy the index vector logical expression are selected, the others are excluded

Example:

`y <- x[x>0]`

y is shorter than x, its elements are the x-elements greater than zero

Arrays and Matrices

An array is a multidimensional object whose elements are all of the same mode (*numeric, logic or character*).

An array is entered with the command **array()** specifying:

- the vector containing the array elements
- a *dimension vector*

A dimension vector is a vector of non-negative integers: if its length is k , the array is k -dimensional ($k=2$, *2-dimensional array or matrix*).

Example:

```
A <- array(1:12, dim=c(3,2,2))
```

or

```
A <- 1:12
```

```
dim(A) <- c(3,2,2)
```

The array is filled with the elements of the vector according to the “**column major order**”, with the first subscript moving faster and the last subscript slowest.

Example:

```
A <- array(1:12, dim=c(3,2,2))
```

	A[, , 1]		A[, , 2]	
	[,1]	[,2]	[,1]	[,2]
[1,]	1	4	7	10
[2,]	2	5	8	11
[3,]	3	6	9	12



Array indexing

- Individual elements of an array can be referenced by giving the name of the array followed by the subscripts in square brackets, separated by commas.

Example:

`A[2,1,1]`

- Subsections of arrays can be created by giving a sequence of index vectors

Example: `select A[2, ,]`

`Sel<-c(A[2,1,1], A[2,2,1], A[2,3,1], A[2,1,2], A[2,2,2], A[2,3,2])`

Example

```
> data(Titanic)
> ls()
[1] "Titanic"
> str(Titanic)
table [1:4, 1:2, 1:2, 1:2] 0 0 35 0 0 0 17 0 118 154 ...
- attr(*, "dimnames")=List of 4
..$ Class   : chr [1:4] "1st" "2nd" "3rd" "Crew"
..$ Sex     : chr [1:2] "Male" "Female"
..$ Age     : chr [1:2] "Child" "Adult"
..$ Survived: chr [1:2] "No" "Yes"
> is.array(Titanic)
[1] TRUE
```

```
> Titanic[,,,]  
, , Age = Child, Survived = No
```

```
Sex  
Class Male Female  
1st 0 0  
2nd 0 0  
3rd 35 17  
Crew 0 0
```

```
, , Age = Adult, Survived = No
```

```
Sex  
Class Male Female  
1st 118 4  
2nd 154 13  
3rd 387 89  
Crew 670 3
```

```
, , Age = Child, Survived = Yes
```

```
Sex  
Class Male Female  
1st 5 1  
2nd 11 13  
3rd 13 14  
Crew 0 0
```

```
, , Age = Adult, Survived = Yes
```

```
Sex  
Class Male Female  
1st 57 140  
2nd 14 80  
3rd 75 76  
Crew 192 20
```

> Titanic[1,,2] #Class=1, Survived=Yes

Age
Sex Child Adult
Male 5 57
Female 1 140

> Titanic[1,,1] #Class=1, Survived=No

Age
Sex Child Adult
Male 0 118
Female 0 4

> Titanic[3,,2] #Class=3, Survived=Yes

Age
Sex Child Adult
Male 13 75
Female 14 76

> Titanic[3,,1] #Class=3, Survived=No

Age
Sex Child Adult
Male 35 387
Female 17 89

Matrices

A matrix is a 2-dimensional array. In R handling matrices is easy and computing time is short.

Arithmetics:

- Element by element operation:
A+B, A-B, A*B, A/B
- Matrix product: A %**% B

Functions:

- nrow() number of rows
 - ncol() number of columns
 - t() transpose
 - det() determinant
 - diag() diagonalize
 - eigen() eigenvalues, -vectors
-

An easy way to build up matrices from vectors or other matrices) is through the functions:

- **rbind()** forms matrices concatenating the arguments by row
- **cbind()** forms matrices concatenating the arguments by column

The arguments can either be vector or matrices with:

- the same number of columns (rbind) or
- the same number of rows (cbind).

Example:

```
> x <- array(1:12,dim=c(3,4))  
> y <- c(0,0,0,0)  
> xy <- rbind(x,y)
```



```
> xy  
  [,1] [,2] [,3] [,4]  
1  1  4  7  10  
2  2  5  8  11  
3  3  6  9  12  
y  0  0  0  0
```

More on R packages

Packages

R has a set of packages that enlarge its potential.

They can be:

- Installed by default and loaded on startup
- Installed by default but not loaded
- Packages to be installed by the user

PACKAGES INSTALLED AND LOADED BY DEFAULT

When you start R, default loaded packages are:

```
> getOption("defaultPackages")
```

```
[1] "datasets" "utils" "grDevices" "graphics" "stats" "methods"  
    (plus, of course, base)
```

PACKAGES INSTALLED BUT NOT LOADED BY DEFAULT

You can get the list of available packages that can be loaded typing:

➤ `library()`

To load packages:

Windows:

Packages -> Load Packages

Linux:

> `library("Package1")`

PACKAGES TO BE INSTALLED BY THE USER

The R website provide links of the CRAN where it is possible to download the “contributed extension packages”

Surprising how many extensions are available!

To install one of them:

Windows:

Packages -> Install packages (chose CRAN or local zip file)

Packages -> Load packages

Linux shell:

R CMD INSTALL “\$path/package1.tar.gz”
